BAB 5
PERANCANGAN DAN IMPLEMENTASI

Langkah-langkah perancangan dan implementasi sistem *K-Shortest Path Routing* pada *Software Defined Network* akan dijelaskan pada bagian ini sesuai dengan yang telah dibahas pada metodologi penelitian.

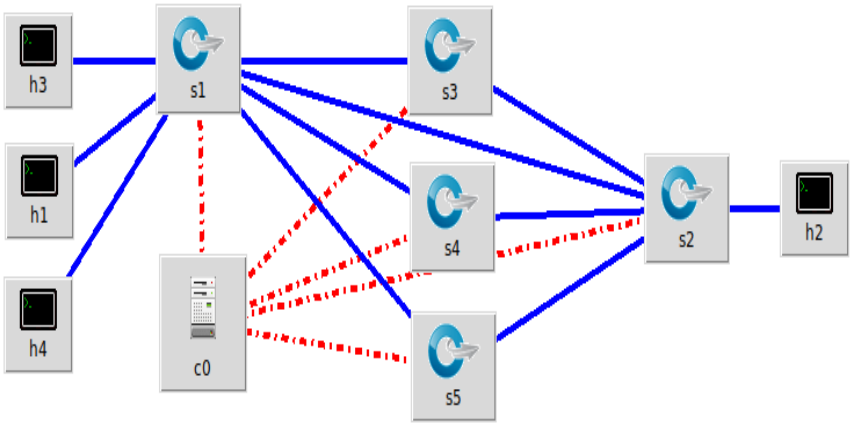
5.1 Perancangan

Perancangan dibutuhkan sebagai tahap perencanaan terhadap sistem yang akan dibangun sesuai dengan bab-bab sebelumnya mengenai persyaratan kebutuhan sistem.

5.1.1 Topologi

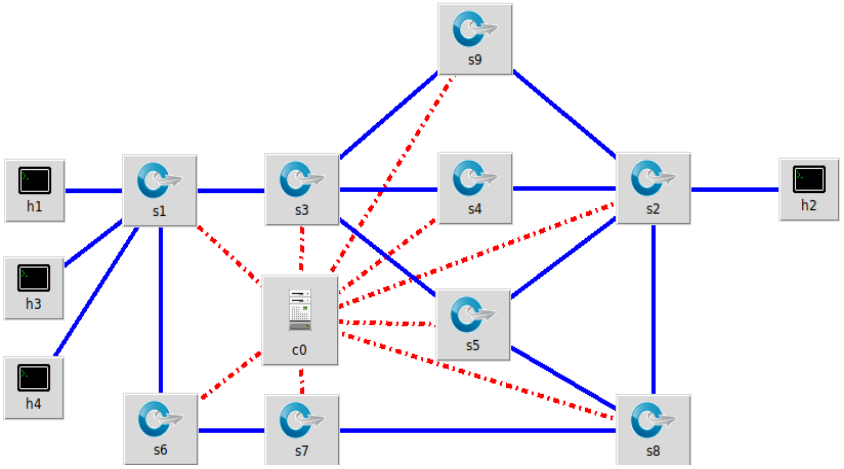
Topologi akan dibentuk sesuai dengan kebutuhan pengujian dari algoritme routing, akan dirancang untuk memiliki banyak *path*, sehingga *controller* memiliki banyak pilihan jalur yang dapat digunakan untuk mengirim paket data.

Untuk pengujian penelitian ini dilakukan pada dua jenis topologi. Hal ini bertujuan untuk mengetahui performa dari algoritme yang telah dikembangkan ke lingkungan jaringan yang berbeda-beda. Seluruh topologi menggunakan *switch* dan *bandwidth* yang sama . Topologi yang digunakan dalam penelitian ini seperti ditunjukkan pada gambar 5.1 dan gambar 5.2



Gambar 5.1 Topologi pertama

Gambar 5.1 merupakan topologi pertama yang digunakan untuk skripsi ini. Topologi ini memiliki 5 buah *switch* dan 4 buah *host.* Pada topologi ini *host* 2 bertindak sebagai *server* dan ketiga *host* lainnya bertindak sebagai *client.* Topologi ini tergolong cukup simple karena hanya memiliki 4 jalur yang berbeda untuk menghubungkan *host* dan *server* yang dapat digunakan untuk mengirimkan data.



Gambar 5.2 Topologi kedua

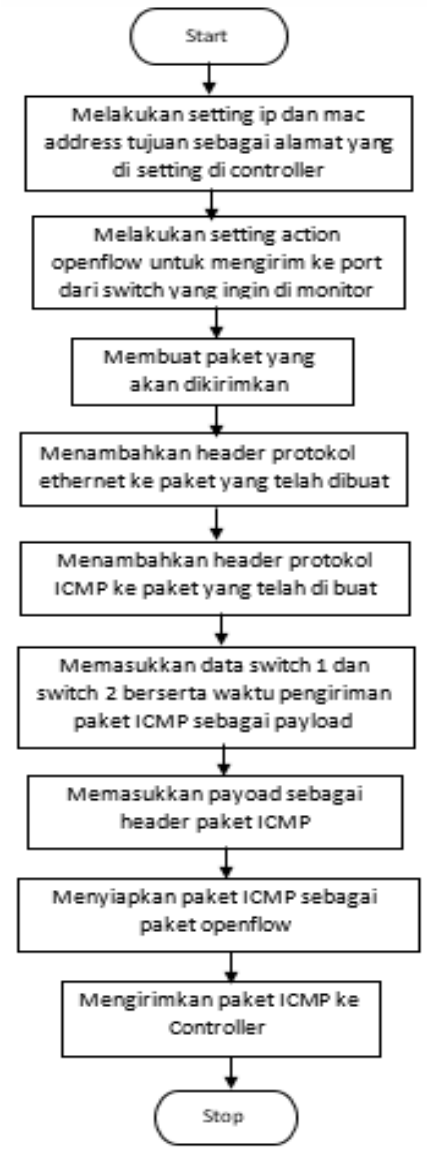
Gambar 5.2 merupakan topologi kedua yang digunakan untuk skripsi ini. Topologi ini memiliki 9 buah *switch* dan 4 buah *host.* Pada topologi ini *host* 2 bertindak sebagai *server* dan ketiga *host* lainnya bertindak sebagai *client.* Topologi ini tergolong cukup rumit karena memiliki banyak jalur sebagai penghubung *host* dan *server* yang dapat digunakan untuk mengirimkan data.

Tabel 5.1 Daftar komponen dalam topologi

No	Ikon	Nama	Fungsi
1		*Host (client&server)*	Berfungsi sebagai *device client/server* pada topologi SDN.
2		*OpenFlow switch*	Berfungsi sebagai *swicth* untuk melakukan *forwading* pada topologi SDN.
3		*Controller*	Berfungsi sebagai *Control Plane* pada SDN.
4		*Link* penghubung *switch* ke *host* atau *switch* ke *switch*	Berfungsi sebagai penghubung antara *switch* dan *switch* atau *switch* dan *host*
5		*Link* penghubung *controller* dan *switch*	Berfungsi sebagai penghubung antara *controller* dan *switch*

5.1.2 Algoritme *Monitoring* Jalur

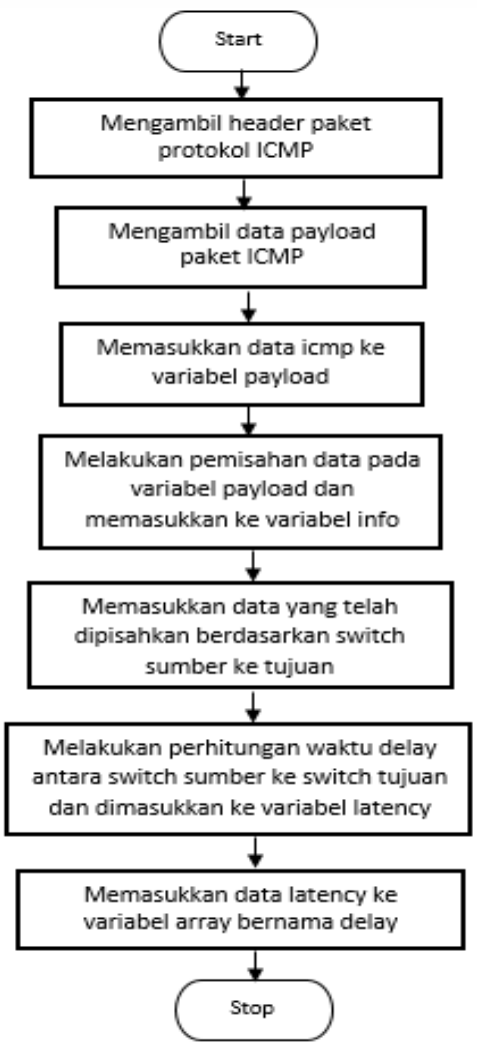
Berdasarkan perancangan topologi dan observasi terhadap sistem yang akan dibangun, algoritme *monitoring* jalur diimplementasikan menggunakan library *ryu controller*. Algoritme ini bertujuan untuk mengukur *delay* antar 2 buah *switch*. *Delay* antar *switch* yang temukan akan ditotal untuk mengetahui *cost* dari sebuah jalur. Terdapat 3 fungsi utama dalam algoritme *monitoring* jalur yaitu fungsi untuk mengirim paket icmp, fungsi setelah paket icmp berhasil diterima, dan yang terakhir fungsi untuk mengambil nilai *cost.* Berikut merupakan *flowchart* dari ketiga fungsi tersebut.



Gambar 5.3 *Flowchart* Fungsi pengiriman paket ICMP

Pertama sistem akan melakukan pengaturan terhadap IP, dan MAC *address* sebagai tujuan dari pengriman paket ICMP, lalu melakukan pengaturan *action openflow* agar sistem mampu mengirimkan paket sesuai dengan *port-port* di

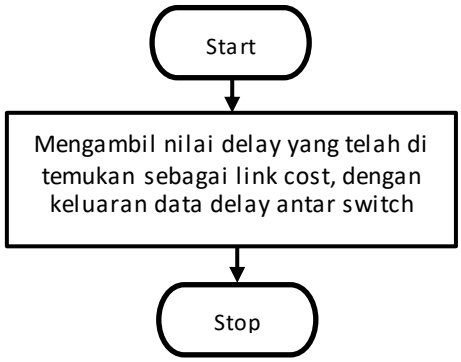
dalam *switch* dalam topologi. Selanjutnya, sistem akan membuat paket untuk dikirimkan, dan *header* yang pertama ditambahkan adalah *header ethernet,* lalu menambahkan *header* ICMP, setelah itu akan memasukkan data dari *switch-switch* yang bertetangga yang akan dilakukan pengiriman data ICMP, beserta waktu pengirimanya sebagai data dalam bentuk *payload*. Selanjutnya sistem akan memasukkan data *payload* sebagai header dalam paket ICMP yang akan dikirim, lalu sebelum paket ICMP dikirimkan, sistem terlebih dahulu akan membuat paket ICMP tersebut menjadi paket *openflow*. *Flowchart* dari fungsi ini ditunjukkan pada gambar 5.3



Gambar 5.4 *Flowchart* Fungsi setelah paket ICMP diterima

Pertama sistem akan melepaskan *header* protokol ICMP yang ada pada paket yang telah diterima, lalu mengambil data *payload* dari paket ICMP tersebut. Selanjutnya data tersebut akan dipindahkan ke variabel bernama payload. Setelah itu sistem akan melakukan *split (*pemisahan) data ke variabel info, yang nantinya akan berbentuk *array.* Selanjutnya sistem akan memasukkan data dalam variabel info sesuai dengan *switch-switch* yang melakukan pengiriman paket ICMP dari

sumber ke tujuan ke variabel S1 dan S2 yang berbentuk *array* juga untuk mengetahui semua *switch-switch* yang bertetangga. Lalu sistem akan melakukan perhitungan waktu yang dibutuhkan oleh paket ICMP untuk sampai dari *switch* sumber ke tujuan dan disimpan dalam variabel latency. Selanjutnya sistem akan memindahkan data latency ke varibel *array* bernama *delay*. *Flowchart* fungsi ini ditunjukkan pada gambar 5.4.



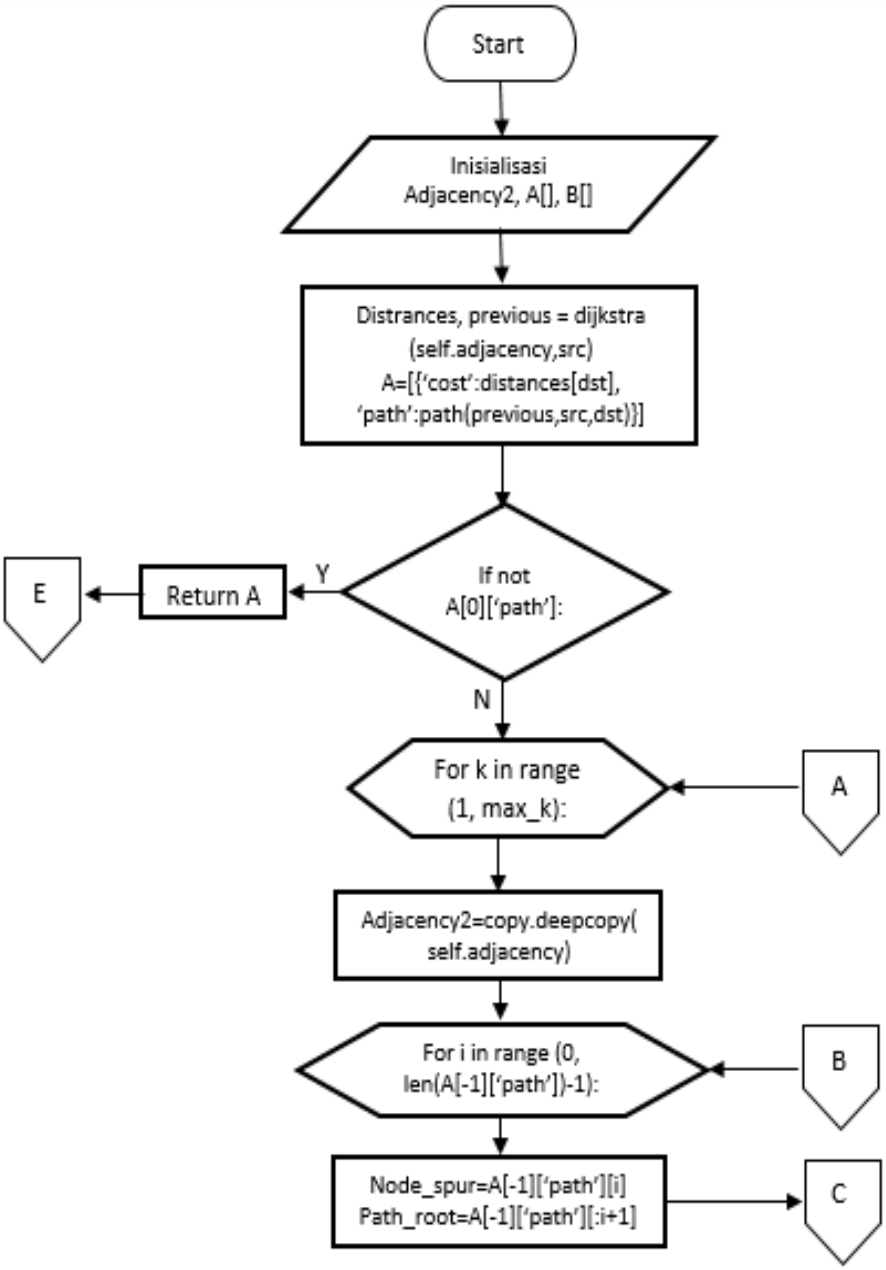
Gambar 5.5 *Flowchart* Fungsi untuk mengambil nilai *cost*

Flowchart fungsi untuk mengambil *cost* sebenarnya hanya fungsi keluaran untuk data *delay* yang telah ditemukan, sistem akan langsung mengambil nilai *delay* yang telah ditemukan pada fungsi sebelumnya dan akan memberikan keluaran data dari variabel *delay* yang berbentuk *array* seperti ditunjukkan pada gambar 5.5. Data ini yang akan digunakan sebagai parameter *cost* dalam pemilihan jalur pada penelitian ini.

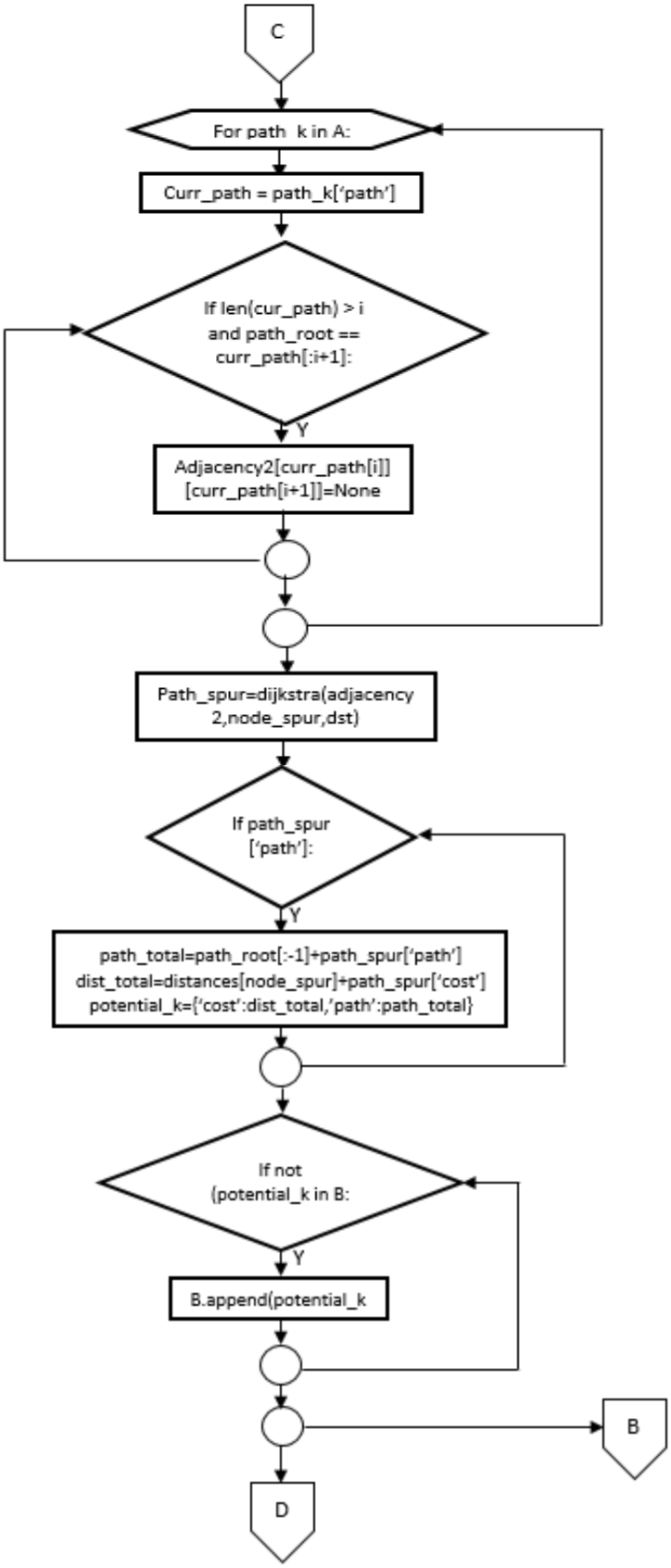
5.1.3 Algoritme Pencarian dan Pemilihan Jalur

Berdasarkan perancangan topologi dan observasi terhadap sistem yang akan dibangun, algoritme pencarian jalur pada penelitian ini menggunakan algoritme Yen. Algoritme Yen akan digunakan untuk mencari jalur terpendek sebanyak K yang dibutuhkan oleh sistem. Source kode untuk algoritme yen dapat dilihat pada gambar 2.3.

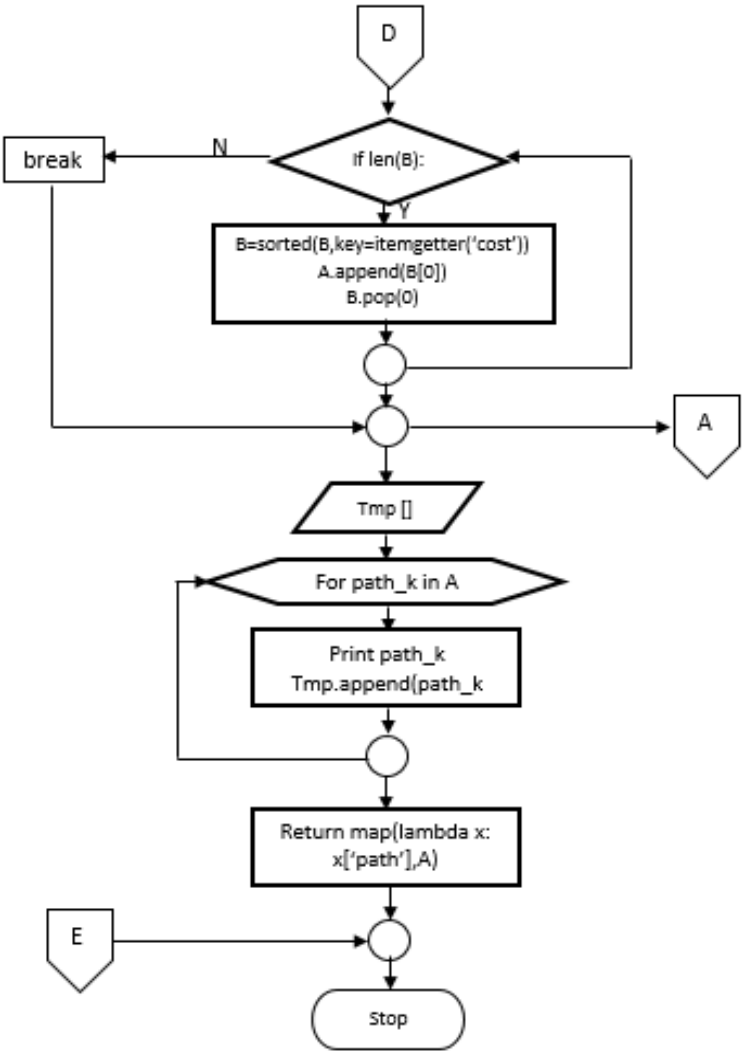
Berdasarkan beberapa jalur yang telah didapatkan dengan algoritme pencarian jalur, maka dilakukan pemilihan jalur dengan mengurutkan jalur dengan *delay* terkecil. Jalur yang memiliki *delay* paling kecil akan digunakan sebagai K-Jalur pengiriman data. Selengkapnya akan ditampilkan pada *flowchrat* pencarian dan pemilihan jalur.


Start
Inisialisasi
Adjacency2, A[], B[]
Distrances, previous = dijkstra
(self.adjacency,src)
A=[{'cost':distances[dst],
'path':path(previous,src,dst)}]
If not
A[0]['path']:
Y
Return A
E
N
For k in range
(1, max_k):
A
Adjacency2=copy.deepcopy(
self.adjacency)
For i in range (0,
len(A[-1]['path'])-1):
B
Node_spur=A[-1]['path'][i]
Path_root=A[-1]['path'][:i+1]
C

(a)


C
For path k in A:
Curr_path = path_k['path']
If len(cur_path) > i
and path_root ==
curr_path[:i+1]:
Y
Adjacency2[curr_path[i]]
[curr_path[i+1]]=None
Path_spur=dijkstra(adjacency
2,node_spur,dst)
If path_spur
['path']:
Y
path_total=path_root[:-1]+path_spur['path']
dist_total=distances[node_spur]+path_spur['cost']
potential_k={'cost':dist_total,'path':path_total}
If not
(potential_k in B:
Y
B.append(potential_k
B
D

(b)



(c)

Gambar 5.6 *Flowchart* algoritme pencarian dan pemilihan jalur

Flowchart algoritme pencarian jalur dan pemilihan jalur pengiriman data ditampilkan pada gambar 5.6 (a)(b)(c). Sistem pertama sekali melakukan inisialisasi variabel-variabel yang dibutuhkan, lalu sistem akan mengambil keluaran dari fungsi Dijkstra dan dimasukkan pada variabel yang berbeda. Data dari variabel Dijkstra ini akan digunakan sebagai *path* pertama dari beberapa jalur yang akan dicari. Setelah itu sistem akan melakukan perulangan untuk mencari jalur lain sebanyak max_k yang telah ditentukan. Setelah sistem sudah selesai menemukan jalur sebanyak K potensial yang dibutuhkan, sistem akan melakukan *sorting* terhadap jalur-jalur yang ditemukan tersebut dan akan diurutkan berdasarkan *cost* yang paling kecil. jalur dengan cost paling kecil akan diambil dan digunakan sebagai *path* yang kedua, fungsi ini akan terus berulah sampai sistem menemukan jalur sebanyak max_k yang sudah ditentukan terlebih dahulu. Setelah semua *path* yang akan digunakan sebagai jalur pengiriman data sudah berhasil didapatkan, sistem akan memasukkan data tersebut ke variabel tmp.

5.2 Implementasi

Tahapan yang dikerjakan pada subbab implementasi mengikuti metodologi penelitian yang telah dibahas, yaitu sebagai berikut.

5.2.1 Instalasi

Instalasi merupakan pemasangan *software* pendukung untuk terlaksananya penelitian yang akan dilakukan. Berikut merupakan *software* pendukung disertai dengan cara instalasinya, yaitu sebagai berikut.

5.2.1.1 Mininet

Mininet adalah lingkungan emulasi jaringan openflow yang merupakan tempat sistem ini dibangun. Instalasi *Mininet* adalah sebagai berikut:

1. Mengambil *source code Mininet* pada *github* resmi *Mininet* dengan perintah:

   ```
   $ git clone git://github.com/mininet/mininet
   ```

2. Menginstall *Mininet* dengan perintah:

   ```
   $ sudo mininet/util/install.sh –a
   ```

5.2.1.2 Ryu Controller

Berikut ini langkah-langkah melakukan instalasi *Ryu* sebagai SDN *Controller*, yaitu sebagai berikut.

1. Menginstall *python-pip* dengan perintah:

   ```
   $ sudo apt-get install python-pip
   ```

2. Menginstall *ryu* dengan perintah:

   ```
   $ sudo pip install ryu
   ```

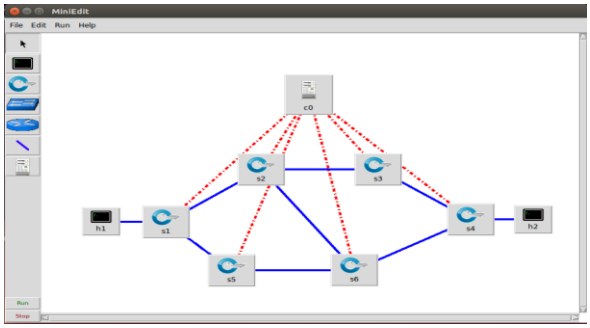
5.2.2 Membangun Topologi di *Mininet*

Setelah semua perangkat lunak pendukung telah terinstall, langkah selanjutnya adalah membangun topologi di *Mininet* sesuai dengan desain topologi yang telah dilakukan. Salah satu cara untuk melakukan pembangunan topologi di *Mininet* adalah dengan menggunakan GUI yang bernama *Miniedit*, yang akan dibahas di bagian ini. Berikut ini langkah-langkah membangun topologi di *Miniedit,* yaitu sebagai berikut.

1. Menjalankan *Miniedit* pada *terminal* dengan perintah berikut:

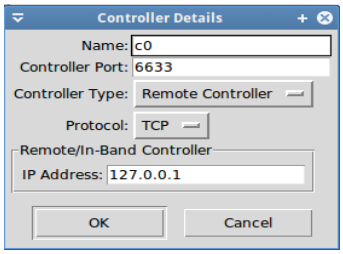
   ```
   $ sudo python  mininet/mininet/examples/miniedit.py
   ```

2. Membangun Topologi. Salah satu contoh hasil pembangunan topologi dapat dilihat pada gambar 5.7.



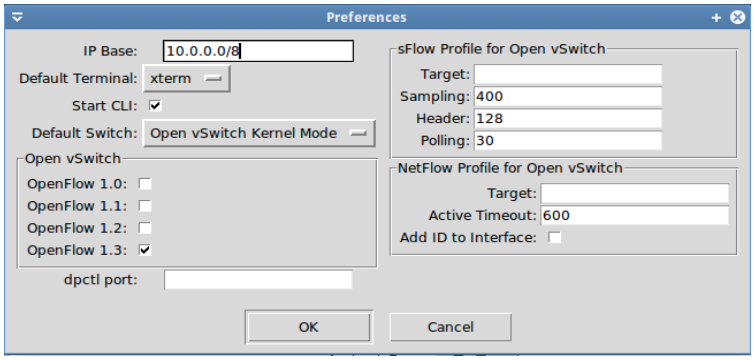
Gambar 5.7 Contoh pembangunan topologi di Mininet

3. Pengaturan *controller* sesuai dengan gambar 5.7 dengan melakukan klik kanan pada logo *controller* (c0) dan menekan "*properties*". ini dilakukan agar Topology yang telah kita kembangkan dapat terhubung ke jaringan Mininet. Pada tab *"Controller Type"* ubah menjadi "*Remote Controller*".



Gambar 5.8 Pengaturan *controller* di Mininet

4. Pengaturan terhadap *preferences* dari Mininet untuk mengaktifkan CLI Mininet dan mengubah versi *OpenFlow* menjadi versi 1.3 sesuai dengan gambar 5.8, hal tersebut dilakukan dengan memilih *menu* "*Edit*" kemudian memilih "*Preferences*" lalu memberikan centang pada openflow 1.3.



Gambar 5.9 Pengaturan *preferences* di Mininet

5. Menjalankan simulasi topologi di Mininet dengan menekan tombol "*Run*".
6. Membuka terminal baru lagi dan menjalankan program *controller* dengan *ryu-manager* menggunakan perintah berikut ini.

```
$ sudo ryu-manager --observe-links nama_program.py
```

5.2.3 Pengembangan Program *Controller*

Pengembangan program *controller* terdiri atas langkah-langkah pemrograman yang dilakukan dalam *Ryu Controller* dengan menggunakan bahasa *Python* versi 2.7.11 dalam mengimplementasi logika atau kecerdasan dari berdasarkan perancangan yang telah dilakukan.

5.2.3.1 *Source Code Monitoring* Jalur

Untuk menemukan *delay* antara dua buah *switch* pada jaringan digunakan *library* dari ryu *controller*. Berikut ini implementasi dari algoritme *monitoring* untuk sistem yang ditunjukkan pada kode listing 5.1.

Kode Listing 5.1 *Source code monitoring* jalur

```
1   def send_ping_packet(self, s1, s2):
2       datapath = self.datapath_list[int(s1.dpid)]
3       dst_mac = self.ping_mac
4       dst_ip = self.ping_ip
5       out_port = s1.port_no
6       actions = [datapath.ofproto_parser.OFPActionOutput(out_port)]
7
8       pkt = packet.Packet()
9               pkt.add_protocol(ethernet.ethernet(ethertype=ether_types.ETH_TYPE_IP,
10                                  src=self.controller_mac,
11                                  dst=dst_mac))
12      pkt.add_protocol(ipv4.ipv4(proto=in_proto.IPPROTO_ICMP,
13                          src=self.controller_ip,
14                          dst=dst_ip))
15      echo_payload = '%s;%s;%f' % (s1.dpid, s2.dpid, time.time())
16      payload = icmp.echo(data=echo_payload)
17      pkt.add_protocol(icmp.icmp(data=payload))
18      pkt.serialize()
19
20      out = datapath.ofproto_parser.OFPPacketOut(
21        datapath=datapath,
22        buffer_id=datapath.ofproto.OFP_NO_BUFFER,
23        data=pkt.data,
24        in_port=datapath.ofproto.OFPP_CONTROLLER,
25        actions=actions
26      )
27
28      datapath.send_msg(out)
29
30   def ping_packet_handler(self, pkt):
31      icmp_packet = pkt.get_protocol(icmp.icmp)
32      echo_payload = icmp_packet.data
33      payload = echo_payload.data
34      info = payload.split(';')
35      s1 = info[0]
36      s2 = info[1]
37      latency = (time.time() - float(info[2])) * 1000  # in ms
38      # print "s%s to s%s latency = %f ms" % (s1, s2, latency)
39      self.delay[int(s1)][int(s2)] = latency
40
41  def get_link_cost(self, s1, s2):
42      return delay[s1][s2]
```

Monitoring link didefinisikan pada kode listing 5.1 dengan menggunakan library *ryu contoller* memanfaatkan protokol icmp dari ping antar *host*. *Delay* yang diambil adalah waktu yang dibutuhkan masing-masing *switch* untuk berkomunikasi. Di dalam pesan icmp tersebut diberikan IP dan MAC *destination* yang merupakan alamat dari *Controller* itu sendiri. Setelah paket tiba di *controller* maka akan dilakukan perhitungan selisih waktu kirim dan waktu setelah paket diterima *controller. Code* diatas akan memberikan keluaran *delay* antar *switch,* yang akan digunakan sebagai *cost* dalam pencarian dan pemilihan jalur.

5.2.3.2 Source Code Pencarian dan Pemilihan Jalur

Untuk menemukan jalur terpendek antara dua buah *host* pada jaringan digunakan algoritme Yen. Berikut ini implementasi dari algoritme tersebut untuk sistem yang ditunjukkan pada kode listing 5.2.

Kode Listing 5.2 *Source code* pencarian dan pemilihan jalur

```
1   def get_paths(self, src,dst,first_port,final_port,max_k=2):
2     adjacency2=defaultdict(lambda:defaultdict(lambda:None))
3     distances, previous = Dijkstra(self.adjacency,src)
4      A = [{'cost': distances[dst],
5            'path': path(previous, src, dst)}]
6      B = []
7      if not A[0]['path']: return A
8      try:
9        for k in range(1, max_k):
10         adjacency2=copy.deepcopy(self.adjacency)
11         for i in range(0, len(A[-1]['path']) - 1):
12           node_spur = A[-1]['path'][i]
13           path_root = A[-1]['path'][:i+1]
14           for path_k in A:
15             curr_path = path_k['path']
16             if len(curr_path) > i and path_root == curr_path[:i+1]:
17               adjacency2[curr_path[i]][curr_path[i+1]]=None
18           path_spur = Dijkstra(adjacency2, node_spur, dst)
19           if path_spur['path']:
20             path_total = path_root[:-1] + path_spur['path']
21             dist_total = distances[node_spur] + path_spur['cost']
22             potential_k = {'cost': dist_total, 'path': path_total}
23             if not (potential_k in B):
24               B.append(potential_k)
25         if len(B):
26           B = sorted(B, key=itemgetter('cost'))
27           A.append(B[0])
28           B.pop(0)
29         else
30           break
31     except:
32       pass
33     tmp=[]
34     for path_k in A:
35       print path_k
36       tmp.append(path_k)
37     return map(lambda x: x['path'], A)
```

Algoritme pencarian jalur dan pemilihan jalur terdapat pada kode listing 5.2. Pencarian jalur dimulai dari *line* ke 2 sampai *line* ke 24. Pertama-tama sistem akan menjalankan algoritme Dijkstra terlebih dahulu untuk mendapatkan *path*

pertama dengan *cost* paling sedikit. Hasil pencarian dengan algoritme *Dijkstra* dibutuhkan sebagai masukan awal dari algoritme K-*Shortest path. Path* yang ditemukan dengan algoritme Dijkstra akan digunakan sebagai *curr-path,* serta node awal dari *path* tersebut akan digunakan sebagai *curr_node.* Setelah itu *a*kan dilakukan pemutusan *link* dari node pertama ke tetangganya untuk mencari *path* baru yang memiliki *cost* paling sedikit secara bergantian. Hal ini dilakukan hingga didapatkan jalur sebanyak max_k. Jika jalur yang ditemukan pada potential_k sudah sebanyak max_k, maka algoritme akan berhenti mencari jalur lain.

Selanjutnya pemilihan jalur dimulai dari line 25 sampe dengan 29. Algoritme akan melakukan *sorting* terhadap jalur yang telah ditemukan untuk mengurutkan jalur berdasarkan *cost* paling kecil dan akan diambil salah satu jalur dengan *cost* paling kecil sebagai *path* kedua. Hal ini akan terus dilakukan sampai didapatkan jalur sebanyak max_k yang akan digunakan untuk pengiriman data.

Algoritme *K-shortest path* diatas akan menghasilkan daftar jalur yang memiliki *de lay* paling kecil sebanyak max_k. Format keluaran data jalur yang ditemukan adalah sebuah *dictionary* yang berisi semua *switch* yang dilalui pada saat pengiriman data secara berurutan. Sebagai contoh, satu jalur antara *switch* 1 dan 5 dituliskan dengan format [1,3,4,5] yang berarti paket data yang melalui jalur tersebut melewati *switch* 1-3-4-5 secara berurutan .